E-949 TN-xxx
August 26, 2005

Experiment 949
Technical Note No. xxx

A new Target Reconstruction and Target Scatter Algorithm

Benji Lewis

Abstract

This technote describes an algorithm *TGrecon.F* for reconstruction of the Target. The code which is largely based upon *swathccd.F*, but is significantly less dependent on information from the UTC track. An algorithm to find kinks in the target track due to a scatter within the charged particle track, *Kinkfinder.F* is also described. The routine *TGconstructor.F* implements the use of *TGrecon* and *KinkFinder*. The structure and performance of these algorithms are shown.

1 Introduction

The reconstruction of the target in nonideal situations, such as Pion scatters, will help in our ability to study these events. In PNN2 the largest background is due to scatters in the target from $K \rightarrow \pi^+ \pi^0$events. Having a true represenation of what occurred in the target may lead to methods of reducing this background and/or be able to measure this background more precisely. In prior analysis (e787,e949-PNN1), the target was reconstructed by *swathccd* or *swath*. This algorithm assumed an ideal track that matches with the UTC's track extrapolation into the target.

Overall, *swathccd* does a good job in determining what happens in the target. *swathccd* has 89.2% of all events reconstructing with pion tracks and 6.4% reconstructing with no pion track (the kaon cluster is at the target edge and near the swath) (about 4.% of all events are not reconstructed). An event that passes *swathccd* does not imply that it was reconstructed correctly, just that it statisfied all of *swathccd*'s requirements. When *swathccd* is unsuccessful in reconstructing the event, stale information is stored in the ntuple common block.

An additional problem with *swathccd* is due to how the code was developed. The non-modularity of the code makes it somewhat difficult to modify without inadvertently affecting other parts of the code. This was due to add-ons and improvements occuring over a long time period. With these known problems in advance, I created an algorithm to perform the target reconstruction, *TGrecon*, as modular as possible so that modifications can be employed with as much ease as is technically acheivable. The main motivation to create an improved *swathccd* was to be able to reconstruct events with a kink in the charge particle track (a scattered event). Also knowing in advance how well *swathccd* already performs on pnn events (99.6% acceptance [1]), I reversed engineered *swathccd* into it's most basic parts and built *TGrecon* in *swathccd*'s likeness with some modifications.

Section 2 will explore *TGrecon*'s overall structure and method of implementation. Comparisons of *TGrecon* and *swathccd* will occur throughout this technote especially where *TGrecon* differs greatly from *swathccd* . To facilitate modifications to *TGrecon* in the future an appendix to this note detailing the uses of the subroutines will be added.

Section 5 will explore *KinkFinder*'s overall implementation and performance in detecting target scatters.

2 TGrecon Algorithm Structure

2.1 Inputs

TGrecon uses as inputs *idctrk, tpiext, Xin, Yin, icharge* which are the same inputs as *swathccd*. UTC track information from track number *idctrk*. *Xin, Yin* is the kaon entering position determined by the B4 detector (values $\geq 8.$ are considered by the algorithm as having no B4 information available). *tpiext* is the expected pion time. *tpiext* = *trs* in `setup_pass1` calls. However, it could be defined differently in cases where the trigger does not require T·2.

- *idctrk* - Index of UTC track
- *tpiext* - Expected pion track time (usually *trs*)
- *Xin, Yin* - x,y value of the entering Kaon from B4 (if $\geq$ 8.0cm, algorithms assume entering position is unknown).
- *icharge* - Charge of the exiting track. Not all routines were created to handle a negative track.

A preburner is called to determine the hits within the target. Hits are fed to *TGrecon* directly from `get_tim_e949new` and `trghits_ccd`.

2.2 Classifying hits

Classifying hits into different particle types is done by time and energy only. The following classifications are done:

Hit Lists	Energy (MeV)	Time (ns)
Kaon	> 4.0	$-8.0 < t < 8.0$
KaonLate	> 4.0	$8.0 < t < 50.0$
Pion	$0.1 < E < 10.0$	$\mid t - trs \mid < 4.0$
Gamma	> 0.1	$-999. < t < 75.0$
Low-E Gamma	$0.05 < E < 1.2$	$-999. < t < 75.0$
mid-E Gamma	$1.2 < E < 4.0$	$-999. < t < 75.0$
High-E Gamma	> 4.0	$-999. < t < 75.0$

Table 1: Candidate hit list. Candidate hits are not joined by geometry, only by energy and time.

As you can tell from the energy and time windows some hits can be in both the pion and kaon/kaonLate hits or vice-versa. When events have delayed coincidence of $< 4.0ns$, some reconstruction problems could occur. In that, kaon hits could to get placed into the final pion track unless careful precautions are taken (and sometimes problems occur even if all precautions are made) . The reduction of *swathccd*'s time window for pions around *trs* from 8ns to 4ns helps somewhat with the misidentification of particles.

2.3 Clustering

Now that we have the hits associated with a particle type we can group these geometrically in the target. We know that a given charge track through the target will 'cluster' together. The problem becomes how do we define a cluster.

The Kaon's track in the target is often refered to a the Kaon blob due to it's pattern in the x-y view of the target. Since the Kaon's movement is mostly along the z direction, in general, the Kaon will traverse a nontrival distance thru all fibers it deposits energy in. So when we cluster the kaon hits we can safely assume that they are no fiber gaps within the Kaon blob. However, the Pion's track is mostly in the x-y direction due to most trigger requirements (KB triggers can have a pion along the z direction of the target, see *Future Improvements*). Also, the Pion deposits less energy per distance than the Kaon. This combination of smaller traversing distance and less energy deposited per distance leaves the possiblity of gaps in the pion track.

The gaps in pion fibers can be due to the pion moving between fibers within the cladding or air gaps. Gaps can cause problems in the reconstruction. For instance, a Gamma converts in the target it will be in time with the Pion and could occur in pion adjacent fibers. This Gamma could be 'absorbed' into the pion cluster even if there is a gap of one fiber between the Gamma and the Pion. This problem can be solved by employing a likelihood function (discussed later). An attempt is made to obtain clusters that have gaps when practical. This idea deviates from *swathccd*.

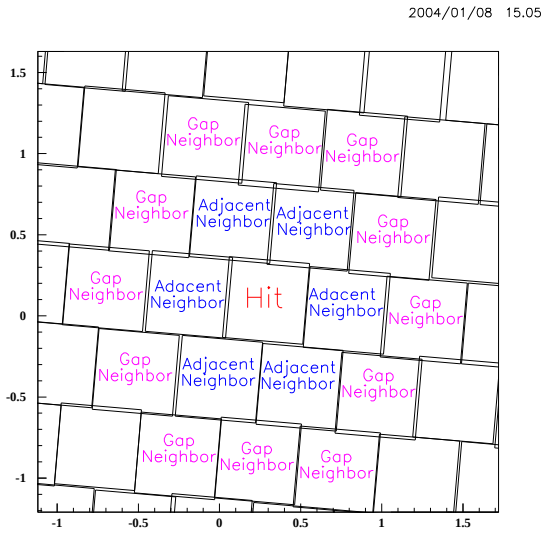


Figure 1: Target fiber neighbors of a hit fiber.

The routine `Cluster_Cand` does the clustering. This routine groups the hits that are adjacent to each other. This routine also allows the ability to group hits that are a gap away from one another. A gap in this case means a neighbor of a neighbor, see Figure 1. Currently pion clusters can have gaps, but kaon clusters are not allowed to have gaps.

A given candidate list can have many clusters, a supercluster. These clusters must be spacially seperated, otherwise they would be considered one cluster instead of many, see Figure 2. Currently, the code allows up to 10 clusters in each supercluster. This is an extension of *swathccd*'s maximum of 5. This change most likely has little to no improvement in the performance of reconstruction.

2.4 Mending

The basic picture of what is happening in the target emerges after the hits are clustered. All kaon or pion hits do not fall within the narrow idea of time and energy from Table 1 and thus the picture of the target is incomplete. The solution to this is to add or *mend* other hits that are close in time

and are neighboring to the existing clusters. It is important to note that the mending process is not an after thought. If these hits were included within the original particle hit list then the clustering could get very messy.

The low, mid, and high-energy gamma list was made for the specific purpose of mending these hits into the clusters. The routine `Mend_fibers` allows the ability to add (from a candidate list) neighboring hits to a supercluster that fall within a specified time window (about the average time of the cluster elements). `Mend_fibers` has been created extremely generalized and so has multiple switches to allow many different mending scenarios as to how it operates see the Appendix but the main switch taken advantage of is allowing (or not allowing) a gapping neighbor (currently only adjacent neighbors are accepted). Another switch allows the removal of the hit from the candidate list. This makes it possible to mend a hit to only one cluster. The bulk of the mending is made in the following order:

- *High-Energy Gamma* with a time window of 0.5ns will mend into the clusters Kaon, K-Late, and Pions.
- *Mid-Energy Gamma* with a time window of 1.2ns will mend into the clusters Kaon, K-Late, and Pions.
- *Low-Energy Gamma* with a time window of 8.0ns will mend into the clusters Pion, Kaon, and K-Late.

The order in which the mending is done is important in that if a hit could statisfy the mending criteria for different supercluster parameters (neighboring and in time) it should go to the supercluster it most likely belongs. A low energy hit most likely belongs to the pion cluster, so the first attempt to mend a low-energy hit will be done with the pion supercluster. Similiarly, the kaon/kaonlate superclusters get first attempt at the high and medium energy hits. The mending order becomes more important when events have small $(T_\pi - T_K)$.

TGrecon uses `Mend_fibers` in the following ways:

- A hit must be within an adjacent fiber, see Figure 2 cluster 3, to the 'core' cluster. The core cluster means original hits within the cluster before the mending begins (`do_grow` $= F$)
- Must be within a given time window which is centered on the average time of the cluster.(see Mended Gamma List above)
- Only allow the candidate hit to be mended into one of the clusters (`do_rem` $= T$)

The `do_grow` switch if turned on will allow the cluster to grow in sometimes unpredictiable ways (especially if the event is noisy), see Figure 2 cluster 4. For more specifics see the Appendix.

After all the mending is complete, a call is made to re-cluster all of the hits within the supercluster. While adding fibers to the superclusters individual clusters my join up to form a bigger cluster. For example, say a given supercluster has 3 (*A,B,C*) seperate clusters and after mending some hits to these clusters, clusters A and B are now just one big cluster while the cluster C is far away from in space from both A and B to join them. So the routine `Re_Cluster` combines cluster A and B into *A'* and C into *B'*. Note that while re-clustering no fibers are added or subtracted.

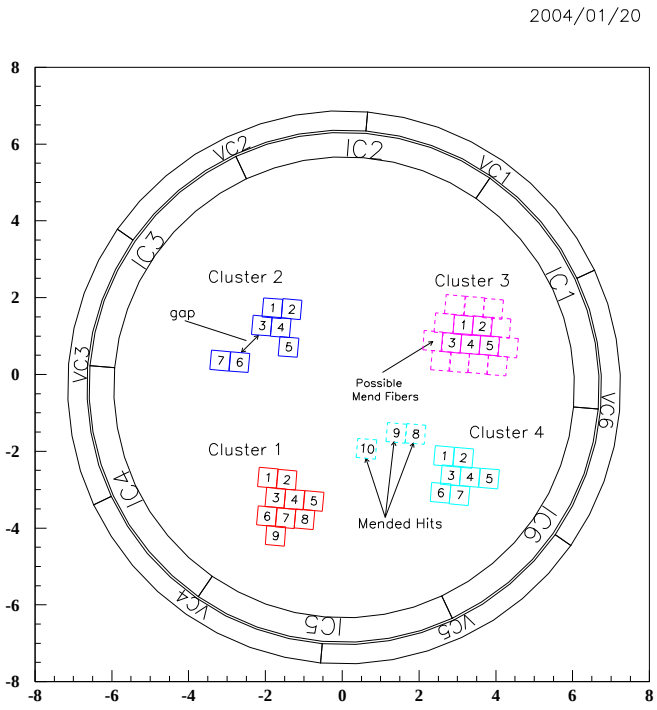


Figure 2: Cluster 1 is a cluster with no gaps. Cluster 2 is a cluster with gaps. Cluster 3 shows the fibers that *TGrecon* allows mending from. Cluster 4 shows how mending could occur if do_grow is TRUE and do_gap is TRUE during mending.

2.5 Likelihood

swathccd applies a likelihood cut to the individual hits within the pion clusters. This is to remove hits that are not pion-like. In this case, pion-like is low-energy hits close the extrapolated UTC track and in time with *trs*. The parameters used to determine this likelihood are time, energy, and distance from the UTC track. If the likelihood function returns a value beyond a chosen threshold (6.0 for *swathccd*) then the hit is kept as a pion. This is a very powerful tool to discern whether or not the hit is a pion or not. However, if the UTC track does not correspond to the path of the Pion then the likelihood function will remove most if not all of the pion hits.

In TG-scatterd events, hits that occurred before the point where the scatter occurs will conceivably fail the likelihood function, due to the large distance from the UTC track. Also, the fiber in which the scatter happens will have some excess energy that will decrease the pion likelihood which will make it appear nonpion-like.

swathccd being highly dependent upon the likelihood function is one reason that it does well at in reconstructing ordinary tracks, but becomes a burden on nonideal tracks. So that *TGrecon* could take advantage of this powerful tool, routines were created to generalize *swathccd*'s likelihood function. `Like_Omit` is a routine will remove nonpion-like fibers from target track in the following ways. (1) If the track does not contain a kink, then the `Like_Omit` is identical to *swathccd* . (2) If a kink is discovered (see *KinkFinder* section), then the track is broken into two segments before and after the scattering point. The hits that occur after the scatter will have the likelihood function applied in the same manner as in *swathccd* . Hits that occurred before the scattering point will apply

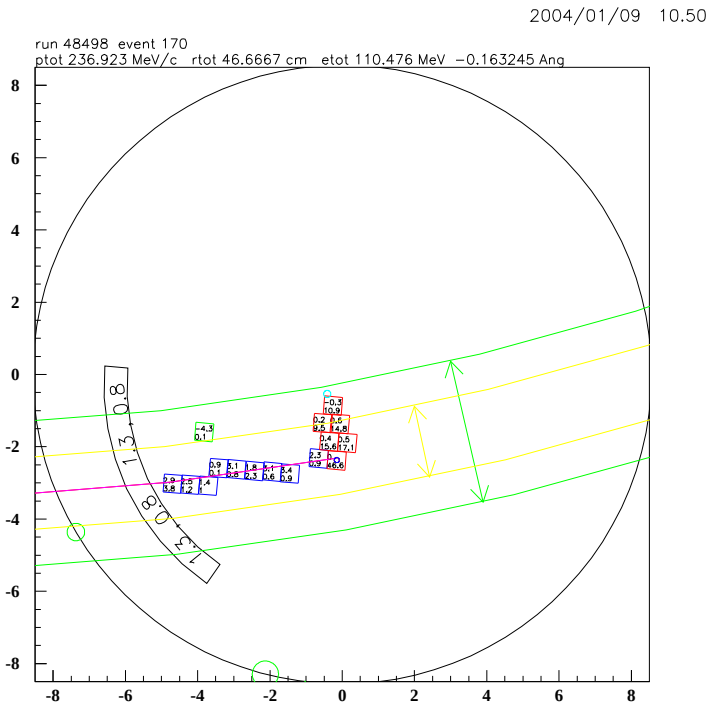


Figure 3: The large green arrow is the size of a 2.0cm swath. The smaller yellow arrow points to a typical sized swath of 1.0cm. Note that the pions are easily contained with the 1.0cm swath

a likelihood function that is not energy dependent (we want to keep high-energy pions that may be from a z-scatter). Therefore, we need track parameters (radius, x,y-center point) that represent the Pion's path before the scatter. This track is calculated in the *KinkFinder* routine, discussed later.

2.6 Matching

At this point the algorithm matches the kaon and pion clusters together. *TGrecon* has 3 different superclusters (Kaon, KaonLate, and Pion). Considering the experiment, we have a Kaon decaying into a Pion within the target. Then the charged particle, pion or muon, traverses the UTC and the Range Stack. So we need to find the pion clusters that 'match' a kaon cluster. Some kaon clusters will be a match geometrically to a pion cluster, but not a match in time, or vice versa. There could be multiple matches such as multiple pion clusters with a single kaon or multiple kaon clusters matching a single pion cluster. So there is a need to create a routine to pick the best match. The algorithm to just this is `Match_Clusters`, a very distinct difference between *TGrecon* and *swathccd* .

2.6.1 Minimum Matching Criteria

- The Kaon enters the target then decays, thus the pion time will naturally occur after Kaon time. We can use this to choose the Kaon-Pi clusters that are matched. Since the pion time is somewhat uncertain, I have allowed a $1ns$ buffer. So that the following is allowed (average pi time) $\geq$ (average kaon time - $1ns$). The average time has no weighting with respect to energy.

- The kaon/kaonlate cluster and the pion cluster must be neighboring each other.

Once the minimum matching criteria is done. The matches are no longer distingished between kaon and kaonlate. So there is no perfered treatment for the kaon clusters (as opposed to the kaonlate clusters). Perfered treatment is done in *swathccd* such that if there is a kaon cluster and a late kaon cluster both on the swath *swathccd* will pick the kaon cluster even if it is on the opposite side of the target from the pion track completely ignoring the late kaon cluster (Jim Frank has modified *swathccd* to reduce this problem).

2.6.2 Picking the Best Match

- If there are no matches, then we have a failure to reconstruct this event.
- If there is only one match, picking the best is trivial.
- If there are multiple matches, then the selection of the 'best' match is attempted.

Now for each match the algorithm determines the distances to the closest tip and whether there are pions on the swath. In general, we are dealing with events that have charged tracks that traverse the UTC and enters the Range Stack. *TGrecon* has been created to make these types of events the priority (see *Future Improvement:No UTC track events* for the other types). With this in mind, the matching algorithm's first attempt at determining the best match is to apply a swath about the UTC extrapolated track, see Figure 3. A check is made to determine if any of the matches have pions within the swath.

- Pion fibers in swath
  - Look for cluster tips that are closest, see Figure 3, i.e. the stopping point of a kaon cluster is near the beginning point of a pion cluster. Progressivly increase what is considered a good cluster-tip match. The values *TGrecon* uses are 0.6,0.9,1.0,1.1,2.0 cm (0.6cm is adjacent fibers, and 0.9, 1.0, and 1.1cm are gap neighbors, 2.0 is a last ditch attempt to find the matching cluster). At the first threshold that has a matching cluster, stop.
  - If there is only one match, we are done. If more than one match exist at this point:

    Then pick the pion cluster that is closest to the exit point.

    If at this point more than one match exist, then the algorithm arbitually picks one. In all likelihood, a perferred match will be found before this could occur.
- No pion fibers in swath.
  - This catagory is considered ($itgqualt = 7, 8$) and is a failure since there is no matching between the UTC and TG.
  - If B4 information is good, then pick the kaon cluster that is closest to the Kaon entering position.

    With the kaon cluster pick, look at matches in the same way as above (match the tips of the clusters and expanded the neighboring area until one match is remaining).

2.7 Calculations

2.7.1 Decay Vertex

The decay vertex is the point in space that the Kaon decayed. Since the spacial resolution in the target is limited by the 0.5×0.5cm square fibers, it is very dificult to determine the exact x-y point other than the decay happened in a particular fiber. With this in mind the decay vertex is always placed at the center of a fiber. In most cases, *swathccd* does this as well. Occasionally *swathccd* pulls the decay vertex to a point very close to the UTC track. *TGrecon* picks the kaon fiber in the following way:

- If the Kaon entering position is known from B4 information, then pick the furthest Kaon fiber from the Kaon-B4 entering position. In some rare case a Kaon may wander in the target enough to make the entering position and the stopping position close. In these cases the decay vertex will be assigned incorrectly.
- If the Kaon entering position from B4 is not known, then `Clust_to_clust_tracking` will pick the Kaon fiber that is associated with the extreme tip of the cluster that is closest to the extreme tip of the matching pion cluster.

2.7.2 Other Quantities

After finding the best pion/kaon cluster match, the rest is calculating physical quantities and filling the *swathccd* common block, *swathccd.cmn*. Since other routines in the `$KOFIA_SOURCE` and `$PASS2_SOURCE` areas have been dependent upon the *swathccd* routine to fill the common block, *TGrecon* is constrained to do the same. In that, all quantities in swathccd.cmn must be filled as *swathccd* intends them to be. An attempt is made to make calculations in the same manner as *swathccd* has done. In some cases slight modifications are done. *TGrecon* calculates the quantities in similiar fashion as *swathccd* has done.

3 TGrecon Performance

The performance of the reconstruction is done by comparing the results of *TGrecon* to the results of *swathccd*. First, we process a data set twice. Once with *TGrecon* reconstructing every event and once with *swathccd* reconstructing every event. Then we only accept events that pass `ps_kink`, $ndclay > 10, tpi-tk > 10$. Also, events cannot have a kinked track, and when *swathccd* reconstructs $itgqualt = 1$ and when *TGrecon* reconstructs $itgqualt = 5 or 6$. The comparison is done on an event by event basis.

The values ploted in Figure 4 are *TGrecon* - *swathccd* . As seen in the log plots, most events have the same TG quantities eventhough reconstructed with different algorithms. This comparison method was the technique that lead to the discovery of *swathccd*'s problem with not choosing late kaon clusters (see *Matching* Section). Also, *swath_bad_check.F* was developed due to the outliers observed.

A visual scan of these outliers show that *TGrecon* is unable to construct an event successfully when there are pion track gaps larger than 1 fiber. This accounts for about one-third of the outlying events. Since *TGrecon* is only called when *swathccd* fails this type of event is unnecessary for *TGrecon* to reconstruct. *TGrecon* was created to allow kinks in the pion track; allowing these types of gaps will reduce the effectiveness of the kink finder. Another deficiency in *TGrecon* is not mending low energy kaon hits into the kaon cluster.

Further scanning of the outliers indicate that *swathccd* still has problems constructing events with late kaon times ($> 8ns$). Approximately one-third of the outliers fall into this catagory. Although, *swathccd* returns `itgqualt` $= 0$ (a successful reconstruction) 75% of these events will be forced to be reconstructed by *TGrecon* due to *swath_bad_check.F*.

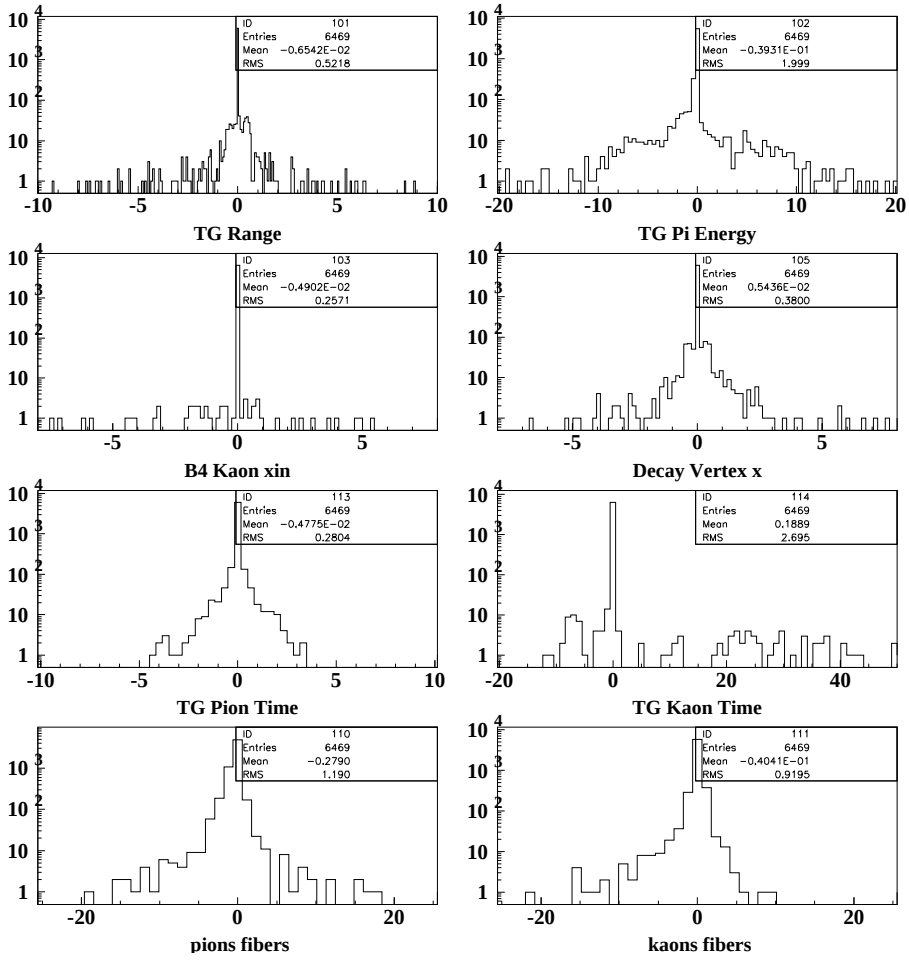


Figure 4: Comparison of *TGrecon* and *swathccd* .

•

4 Additional & Future Improvements to TGrecon

4.1 Swathccd checking routine

In some cases, *swathccd* will reconstruct an event incorrectly. It was determined that the total kinked sample could be increased by about 64% by determining which events were incorrectly reconstructed by *swathccd* . The routine *swath_bad_check.F* was created to look at events that *swathccd* reconstructed to determine if they should be forced to fail and so *TGrecon* could have an attempt to reconstruct the target successfully. Note that this routine is not an improvement to *TGrecon*, but instead improves *swathccd*'s ability to determine if the target was correctly reconstructed.

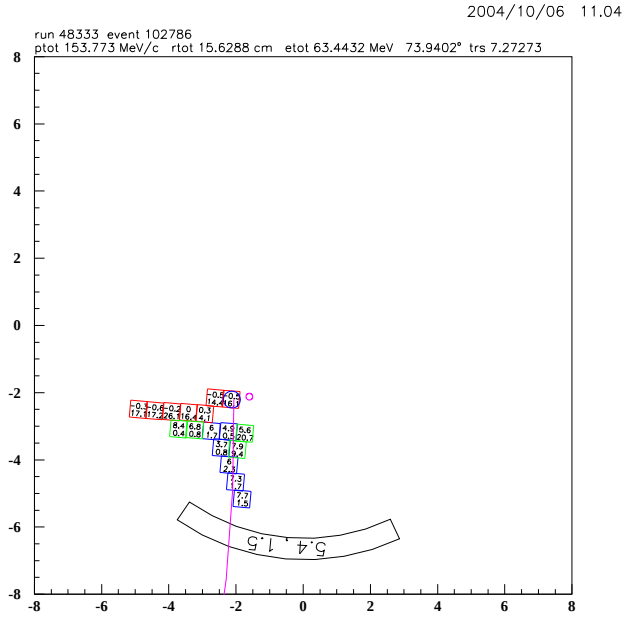
The classes of events (see the following event reconstructions - leftside: *swathccd* , rightside: *TGrecon.*) that *swath_bad_check.F* tries to recover are the following: (1) The fiber in which the scattering occurred has large energy and *swathccd* assigns the scattering fiber as the Kaon. (2) The geometry is such that part of the kaon blob lies within the swath eventhough the Kaon decay occurred outside the swath region. (3) A subclass of (2) is when *swathccd* picks a Kaon that has a large Kaon/Pion gap. (4) *swathccd* assigns high energy pion fibers (presumably from a z-scattered pion) that is out of time with real Kaon as kaon fibers. The result is a kaon blob with a large spread in the times of the fibers.

Class (1) is recovered by making any events with a DELCO of $\leq 3.0ns$ to be reconstructed by *TGrecon.* Class (2) and (3) are principally recovered by forcing a *swathccd* failure on events with a Kaon/Pion gap of greater than 0.75cm. Class (2) events are also recovered by observing where the entering position of the Kaon in the target from B4 information and *swathccd*'s measurement of where the decay vertex occurred. In effect, we attempt to recover events where the decay vertex is grossly mismeasured. Class (4) are recovered by determining if there are mis-identified pion fibers that are labeled as kaon fibers. This is accomplished by observing some *swathccd* 'kaon' fibers that are intime with *trs* and some fibers that are intime with B4.

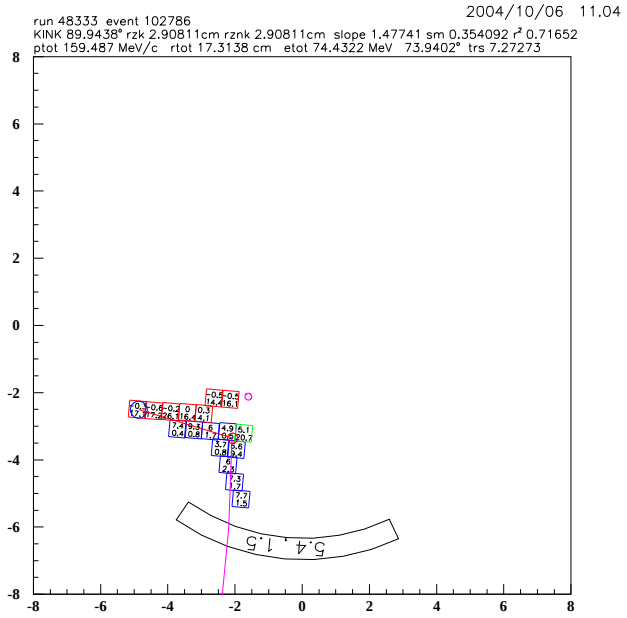
swath_bad_check.F was able to force all kinked events that were found to be wrongly assigned as a *swathccd* success. With the use of *swath_bad_check.F*, the total sample of target scatters would grow by approximately 59k events (78M pass1 events).

PNN1/PNN2 sample	# events
Total	73007
ps_kink	6829
flagged kinks	86
flagged as *swathccd* mistake	55
recovered from swath_bad_check	55

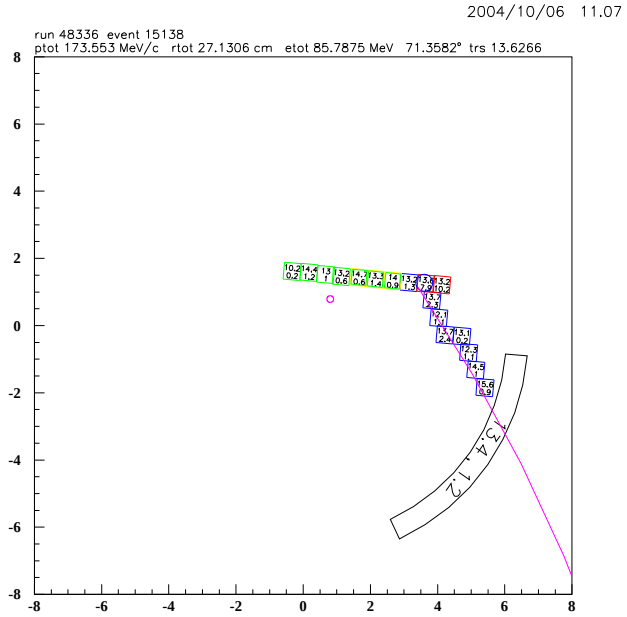
Table 2: *swathccd* checking routine. Notice that all events that were visually identified as a *swathccd* mistake were able to be recovered.



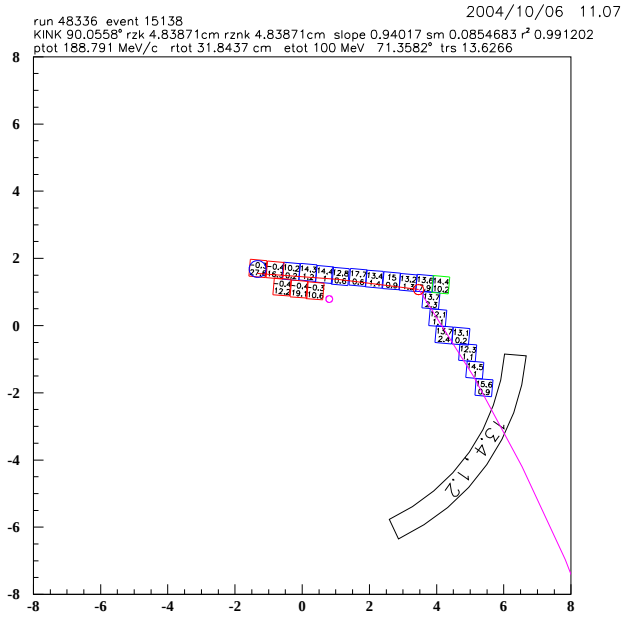
Class (2) *swathccd*'s reconstruction



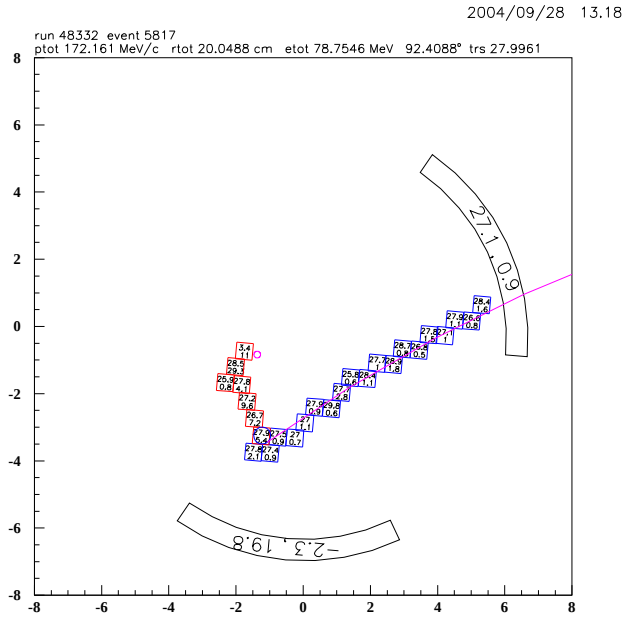
Class (2) *TGrecon*'s reconstruction



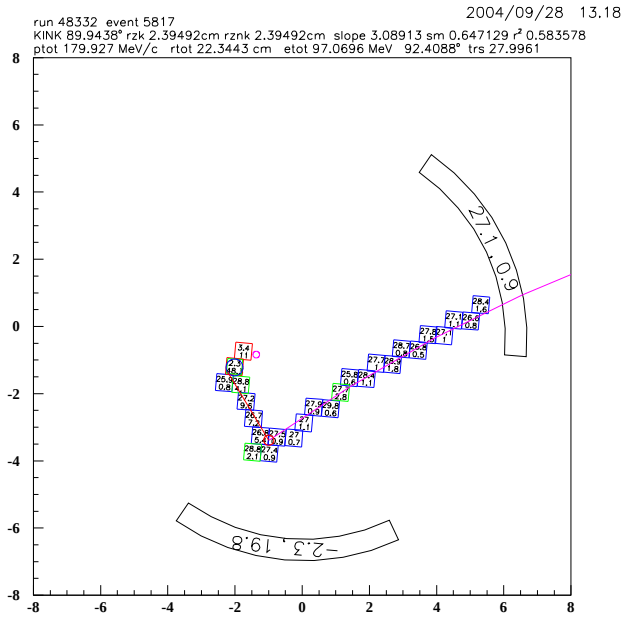
Class (3) *swathccd*'s reconstruction



Class (3) *TGrecon*'s reconstruction



Class (4) *swathccd*'s reconstruction



Class (4) *TGrecon*'s reconstruction

4.2 Kaon decay at TG edge (*itgqualt*=1)

TGrecon does not handle events where the Kaon decayed at the edge of the target in such a way they are no pion fibers. *swathccd* handles these events well. Implementing this type of event in *TGrecon* is needed for a complete break-away from *swathccd* . Otherwise, *TGrecon* would have to be a secondary call within *swathccd* . Since the purpose of creating *TGrecon* was to allow kinks in the pion track, no effort was put into an event where the Kaon decayed at the edge.

4.3 Decays in the Z-direction

In KB triggers it is possible to have an event were the charged product of the Kaon decay is in the z-direction and so does not enter the Drift Chamber. Therefore the time of the decay is unknown. Currently, *TGrecon* cannot handle this type of event correctly since pion hits are classified initially by timing from *trs*. However, with a some minimal effort one could input exteral times into *TGrecon* until a good match is made. It is likely that some of these events are in the class of *itgqualt*=7,8. However, it would require that the accidental track that triggered the $T \cdot 2$ would be intime with the actual decay.

5 Kinkfinder Algorithm Structure

The algorithm *KinkFinder* was created to find charged tracks that scatter in the target. Since the pion is primarily the charged particle that will scatter in the target most references to the pion track is refering to any charged decay product of the Kaon. *KinkFinder* was made to be independent of *swathccd* or *TGrecon*. *KinkFinder*'s ability to find a kink in the track first assumes a good fit from the UTC. If the wrong UTC hits are used during the fitting or the UTC fitting routine chooses the wrong track, then *KinkFinder* will fail to truely determine if there is a scatter in the event. Unlike *TGrecon*, *KinkFinder* was completely built from scratch and as such there are comparisions to

previous routines. Various techniques were used to test the reliablity of *KinkFinder* by using events that were reconstructed without observing a scatter. These studies can be seen in the *Performance* section.

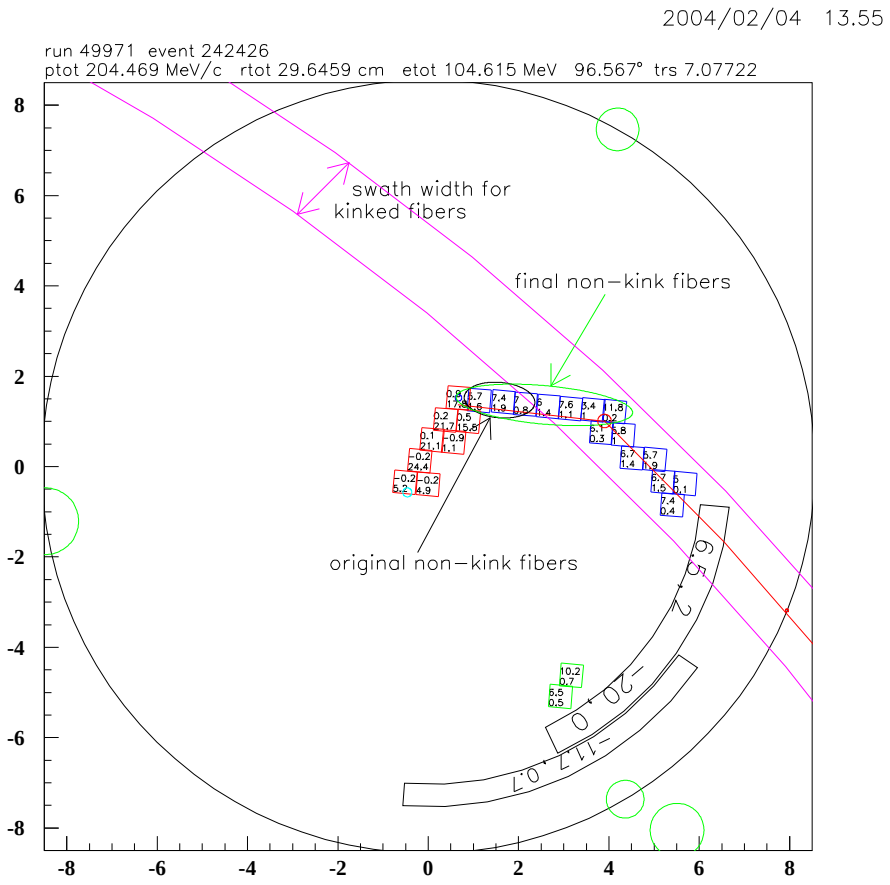


Figure 5: *KinkFinder* Technique

5.1 Inputs

KinkFinder is dependent upon a single cluster (picked from a supercluster), a UTC track (*idctrk*), and the decay vertex of the Kaon.

5.2 Defining kinked fibers

As seen in Figure 5, kinked fibers are the fibers that occur after the pion has scattered. As such, nonkinked fibers are the pion fibers from the decay vertex to the scattered point. The fiber in which the scatter occurs is initially defined as part of the kinked fibers since the fiber should lie on the UTC track. The *KinkFinder* algorithm determines the kinked fibers as the fibers that are within a swath of the UTC track. The current swath value is 0.8cm from the UTC track.

5.3 Minimum Criteria

To remove possible false positives (identified as a scattered event by the algorithm, but is in fact a nonscattered event), some criteria have been placed on the suitiablity of the event as a kink.

- There must be at least one fiber near the exit point of the target.
- There must be at least 2 nonkinked fibers.

5.4 Slope Calculation

Many way could be used to identify if the fibers are diverging away from the UTC track. The method employed two values describing the nonkinked fibers. (1) Distance of the arc length along the UTC track to the nonkinked fiber. (2) The distance of closest approach from the nonkinked fiber to the UTC track. Now we appply a least-squares fit to (2) vs. (1). To be considered a kink the slope must be ≥ 0.1.

If the event does not pass the items in *Minimum Criteria* or pass the slope threshold cut, then it we do not consider the event to have a scatter. If the event passes all of the cuts so far, then the algorithm can procede.

5.5 Finding a Nonkinked Trajectory

To procede further, the nonkinked hits are sent into a modified version of *Toshio's Target Fitter*. For the nonkinked track the fitter gives *KinkFinder* information from a track circle (radius, center point). We do not expect all nonkinked fibers were originally labeled as such. So after the fitter returns parameters for a track, we use this track to determine if other hits exist that lie with a swath around the fitted track. Additional hits are added one at a time and must be close to the previous nonkinked section of the track. With the added hit included *Toshio's Target Fitter* is applied again. This process is repeated three times. One fiber is added at a time to build the nonkinked portion of the track in a predictable and precise way. The three added fiber limit is to prevent the nonkink track 'absorbing' all the kinked hits in a small angle scatter situation. In small angle scatters it could be difficult for an algorithm to pick whether the hit should be associated with the nonkinked or kinked portion since both the nonkinked (fitted) track and the utc extrapolated track could be traversing many of the same fibers.

6 Kinkfinder Performance

To gauge the performance of the kink finder, we forced the UTC track to be rotated about a point on the track and close to the target edge, see Figure 6. To allow Km2 events with a good reconstruction in the UTC and a clean target the following cuts were applied to a Km21 sample: $220 >$ `ptot` $> 260, $ `npi_tg` $< 3, $ `ndclay` $< 10, $ `itgqualt` ≥ 1, `pscut02_kink` cut. The efficiecny was measured at one degree increments between a UTC rotation of -45 and +45. The sample used here is contrived, but it gives an idea of how well the kink finder's angle threshold works. The fact that the UTC expolated track does not lie on the target track due to energy loss could lead to an appearance of a small angle kink lead to the use of the angle threshold. The angle threshold limits the minimum angle to be approximately 5.7^o, see the Slope Calculation section. As seen in Figure 7, the efficeny at large angles is very high.

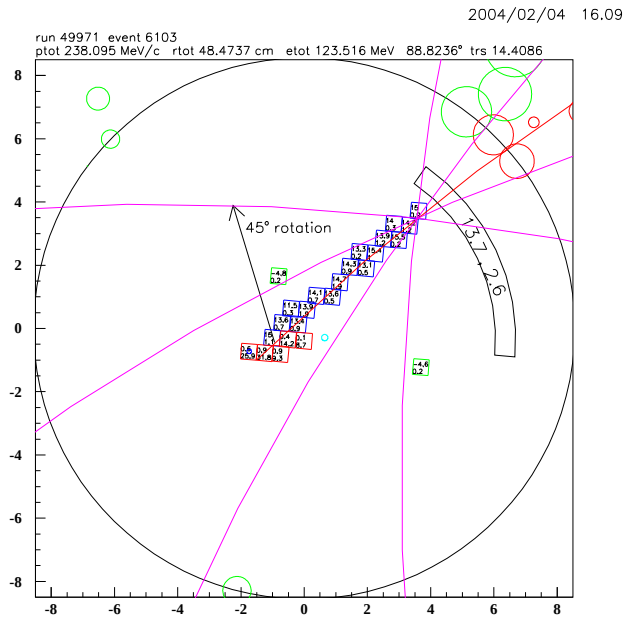


Figure 6: Forced Kink. The UTC track is rotated about a point on the track near the edge of the target

To determine how often the kink finder flags an event as a kink when in fact there were no kink in the track, false positive. We take a sample of Km21 events, which we do not expect many scatters to occur, and apply the following cuts: `ps_kink`, `ndclay` $< 10, 200 >$ `pdc` > 240, `rrs` < 35, `ers` < 80. These cuts are applied as to not involve variables from the target that may be affected by the results of the kink finder output. The `ndclay` cut is to force a good track in the UTC and cuts on `pdc`, `rrs`, `ers` are used to select Km2 events. The false kink rate is determined by visually scanning for real scatters in the sample of events that pass the setup cuts and are flagged as kinks. This will give us the total number of falsely identified kinks. This yields a false kink rate of 0.015%. So the number of false positives are negligible. However, these are for events which are clean in the target.

Photon conversion in the target in Kp2 events could lead to a misreconstruction which could 'confuse' the kink finding algorithm. Hence in Kp2 events we would expect a larger false kink rate than in Km2 events. Since the number of kinks in Kp2 is much larger it is difficult to select a false kink sample. Therefore, visual scans of Pass1 Output events (PNN1/PNN2 triggers) flagged as scatters were done. `ps_kink` was the only setup cut applied. Out of 120 events, 114 were visually identified as a kink (6 of these were a IC scatter) and 6 were not kinks. Of the 6 falsely flagged kinks 5 had a poor UTC track fit. The other event failed due to the decay vertex being assigned incorrectly. The 5 poor UTC track events had `ndclay` ≤ 9. *KinkFinder*'s dependance upon the extropolated UTC track suggest all studies involving kinks should include UTC qualities cuts. So with appropriate UTC cuts we should expect the false kink rate in PNN1/PNN2 triggers to be at most 1.%.

Table 4 is a summary of a PNN1 and PNN2 trigger sample. 3.0% of PNN2 Boxed events are flagged as a kink and only 0.24% of the PNN2 Boxed events pass `ps_kink` cut and are kinks. Using the post-`ps_cut` rate and having 78M total events, we would expect 29k kinked events in the PNN2 Box and 160k kinked events in the entire e949 data set.

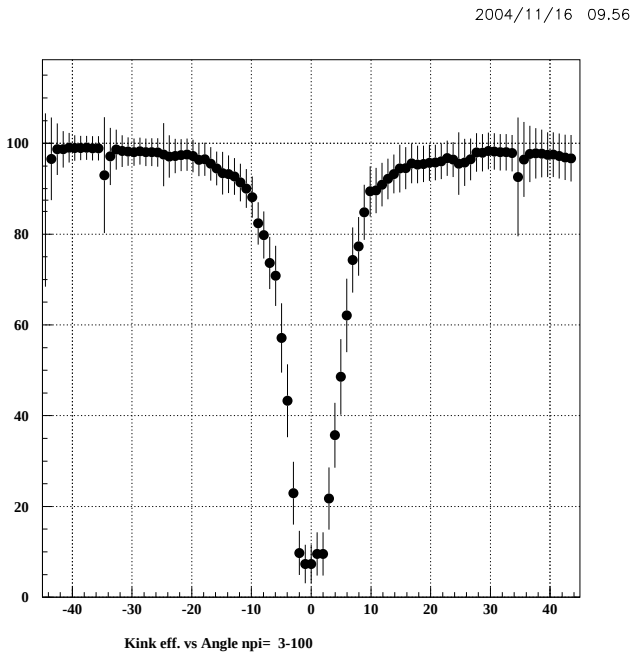


Figure 7: Efficiency vs. Angle of Forced Kinks. The UTC track is rotated about a point on the track near the edge of the target for Km21 events.

	Total	Kinks	Rate
Km21 sample	384115	2250	0.586%
After cuts	159861	67	0.042%
Visual scan	67	43	-
	Total	False kinks	False Rate
False kink rate	159861	24	0.0150%

Table 3: Km21 False Kink Rate.

7 TGconstructor.F

To fully implement the Likelihood function discussed in section 2.5 it was necessary to place *TGrecon* and *KinkFinder* in a larger construct. This routine is called *TGconstructor.F* and is the routine that is called when *swathccd* fails. Basically, *TGconstructor* calls *TGrecon* to reconstruct the target hits and then orders the pion fibers. While ordering the pion fibers there is a search for a scattering point and opposite-side pions(hits ajacent to the Kaon blob but on the opposite side with respect to the exiting track). At this point, the scatter point is given to *KinkFinder* to determine if the event has a kink in the target track. The likelihood function is now applied in the manner prescribed in section 2.5. Since some fibers may have been removed by the likelyhood function, *KinkFinder* is called once more.

	No cuts		After ps_kink	
	All	Kinks	All	Kinks
N	80319	3534	8737	166
PNN2 Box	12581	377	447	30
Kpi2 Box	10573	349	3181	73

Table 4: Target Kink Sample

8 Conclusions

In conjunction *TGrecon* and *KinkFinder* are able to reconstruct target scattered events successfully. After applying `ps_kink` as a setup cut, we expect a yield of 160k scattered events in the e949 sample with 29k within the PNN2 box. Additional cuts such as UTC quality cuts should also be applied to futher remove any misidentified kinks in the sample.

References

[1] "PNN2 1996 1/3 analysis", TN-385, M.V. Diwan, July 21, 2001.